廣東工業大學

操作系统课程设计

学　　院　计算机学院

专　　业　网络工程

年级班别　网络工程 18（4）班

学　　号　3118005320

学生姓名　陈智超

2020 年 7 月

一、课程设计目的

操作系统是计算机系统配置的基本软件之一。它在整个计算机系统软件中占有中心地位。其作用是对计算机系统进行统一的调度和管理，提供各种强有力的系统服务，为用户创造既灵活又方便的使用环境。本课程是计算机及应用专业的一门专业主干课和必修课。

通过课程设计,使学生掌握操作系统的基本概念、设计原理及实施技术,具有分析操作系统和设计、实现、开发实际操作系统的能力。

二、课程设计内容和要求

- 提交一批作业（>=10），按先来先服选择一部分作业（最多 5 个）进入内存
- 为每个作业创建一个进程，并分配内存（用户内存：0—1024K，采用可变连续分配方式）
- 进程调度功能（时间片轮转）
- 随机阻塞进程，并在一段时间后唤醒进程（选做）
- 显示相关信息：后备作业队列、内存分配情况、进程信息、完成作业情况
- 这些功能要有机地连接起来

三、软、硬件环境

软件：Windows 10, JDK1.8，IntelliJ IDEA

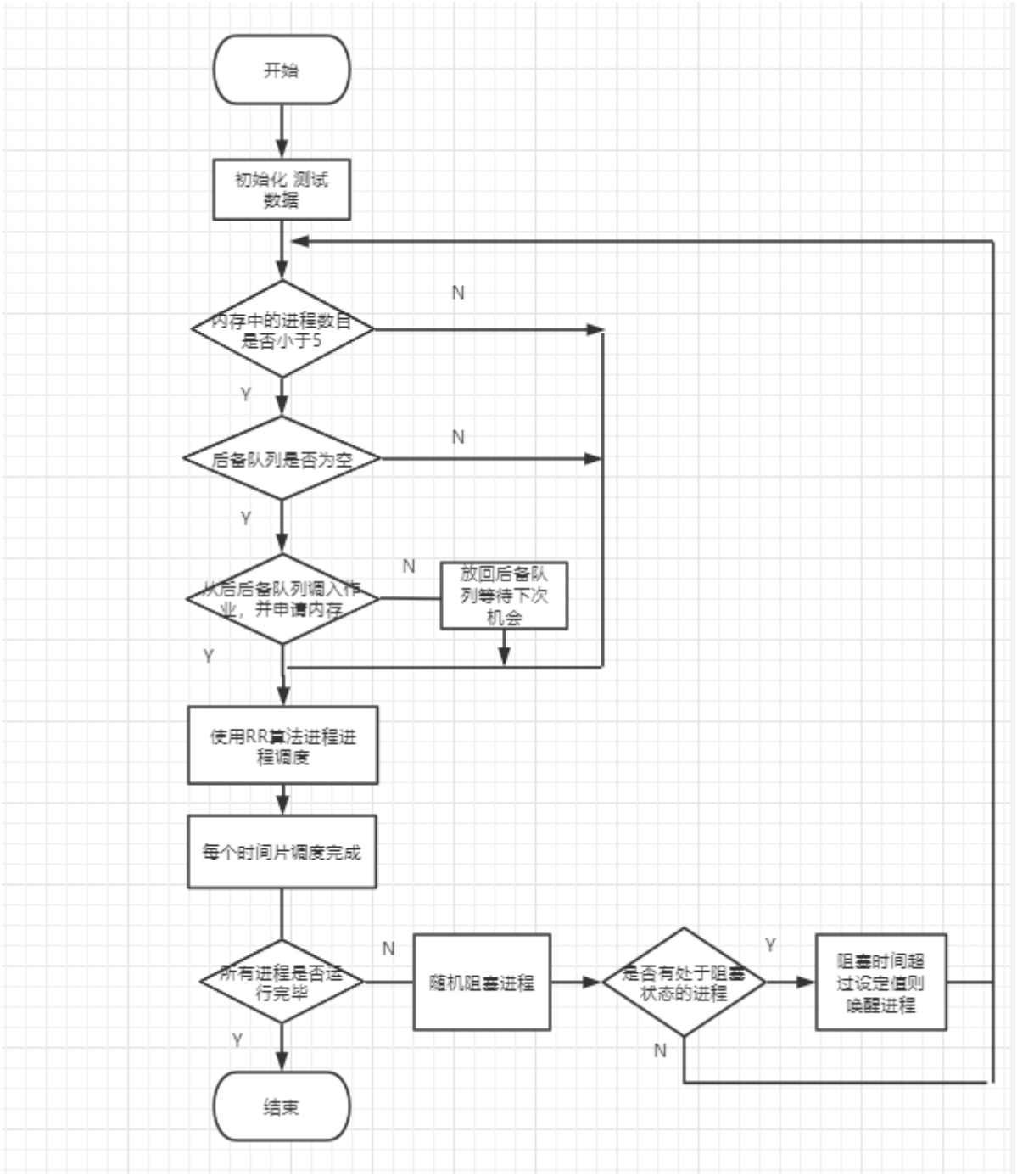
硬件：Intel(R) Core(TM) i5-8300H , 2.30GHz, RAM: 8.00GB

四、设计步骤

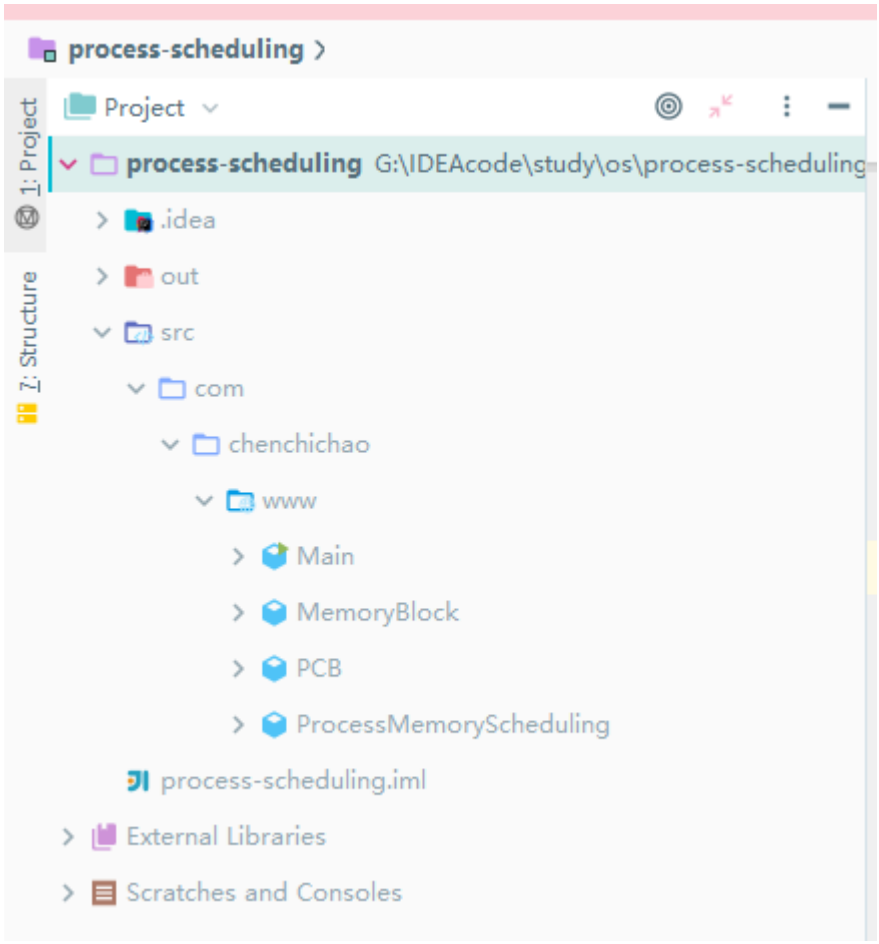
1. 题目分析

本次课程设计是将作业调度，内存管理、进程调度、进程阻塞等功能有机结合起来的一道题目。首先，需要使用随机数初始化 10 个作业，放入后备队列中，然后使用先来先服务（FCFS）进行作业调度，使用时间片轮转算法进行进程调度。其中，最多只能有五个作业能同时进入内存，本实验假设阻塞状态的进程依然在内存中。也就是说，处于就绪、运行、阻塞三种状态的进程数目之和最多为 5 个，即并发进程数最多为 5 个，在进程结束后，就会被调出内存，使用 FCFS 算法从后备队列中调入新的作业。在内存中的几个非阻塞状态的进程使用时间片轮转（RR）算法进行调度。而作业在进入内存之前，是要申请内存的，这时使用首次适应（FF）算法申请内存，从空闲分区链中找到合适的空闲分区并分配给该进程。在进程结束时，要回收其占用的内存，并进行相应的空闲分区合并。如此，便可以将作业调度、内存管理、进程调度和进程阻塞与唤醒这几个功能有机地连接起来了。

2.算法流程图：



3.项目工程截图：



4. 主要的数据结构：

```
// PCB 类
public class PCB {
    // 进程名
    String name;
    // 所需内存
    int needMemory;
    // 主存起始位置
    int address;
    // 到达时间
    double arriveTime;
    // 需要运行时间
    double needTime;
    // 已用时间
    double hasUsedTime;
    // 进程状态：运行中、就绪、阻塞(Running、 Waiting、 Blocking)
    String status;
}
```

```
// 内存块类
public class MemoryBlock {
	int address = 0;
	int length = 0;

	// Busy 或 Free
	String status = "Free";

	public MemoryBlock(int address, int length) {
		this.address = address;
		this.length = length;
	}
}
```

5. 主要算法和代码：

```
// 快速排序，先来先服务中用快速排序算法对作业进行排序
public void quickSortByArriTime(int low, int high) {
		//开始默认基准为 low
		if (low < high) {
			//分段位置下标
			int standard = getStandard( low, high);
			//递归调用排序
			//左边排序
			quickSortByArriTime( low, standard - 1);
			//右边排序
			quickSortByArriTime(standard + 1, high);
		}
}
```

```
  /**
	* 申请内存的方法
	* @param curr 进程的索引（下标）
	* @return
	*/
 public int applyMemory(int curr) {
		if (pcb[curr].status != "U") {
```

```
            return -2;
        }

        MemoryBlock target = null;
        int needMemoryry = pcb[curr].needMemory;
        for (MemoryBlock memoryBlock : memoryBlockList) {
            if (memoryBlock.status.equals("Free")
           && memoryBlock.length >= needMemoryry) {
                target = memoryBlock;
                break;
            }
        }
        if (target == null) {//找不到合适的内存块用于分配
            return -2;
        } else if (target.length == needMemoryry) {
            // 申请的内存和内存分区刚好相等，直接将整块分区分给该进程
            target.status = "Busy";
            return target.address;
        } else {
            target.status = "Busy";
            MemoryBlock block = new MemoryBlock(target.address + needMemoryry,
                                target.length - needMemoryry);
            target.length -= block.length;
            // 将新建的空闲分区插入到分区连中
            memoryBlockList.add(memoryBlockList.indexOf(target) + 1, block);
            return target.address;

        }
    }

    // 释放内存分方法
      public void releaseMemory(int address) {
            MemoryBlock target = null;
            for (MemoryBlock memoryBlock : memoryBlockList) {
                if (memoryBlock.address == address) {
                    target = memoryBlock;
                    break;
                }
            }
            if (target == null){
                return;
            }

            if(memoryBlockList.size()==1){
```

```
            target.status = "Free";
            return;
        }


        int index = memoryBlockList.indexOf(target);

        if (index == 0) {
            MemoryBlock memoryBlock = memoryBlockList.get(1);
            if (memoryBlock.status.equals( "Free")) {
                target.length += memoryBlock.length;
                memoryBlockList.remove(memoryBlock);
            }
            target.status = "Free";
        } else if (index == memoryBlockList.size() - 1) {
            MemoryBlock m = memoryBlockList.get(index - 1);
            if (m.status.equals("Free") ) {
                m.length += target.length;
                memoryBlockList.remove(target);
            }
            target.status = "Free";
        } else {
            MemoryBlock preMemoryBlock= memoryBlockList.get(index - 1);
            MemoryBlock nextMemoryBlock = memoryBlockList.get(index + 1);
            if (preMemoryBlock.status.equals("Free")
                    && nextMemoryBlock.status.equals( "Free")) {
                preMemoryBlock.length += target.length;
                preMemoryBlock.length += nextMemoryBlock.length;
                memoryBlockList.remove(target);
                memoryBlockList.remove(nextMemoryBlock);
            } else if (preMemoryBlock.status.equals("Free")
                        && nextMemoryBlock.status.equals("Busy") ) {
                preMemoryBlock.length += target.length;
                memoryBlockList.remove(target);
            } else if (preMemoryBlock.status.equals("Busy")
                        && nextMemoryBlock.status.equals("Free") ) {
                target.length += nextMemoryBlock.length;
                target.status = "Free";
                memoryBlockList.remove(nextMemoryBlock);
            } else {
                target.status = "Free";
            }
        }
    }
```

```
// 从后备队列中选择作业并申请内存
public void getNextOnDisk(double allTime) {
    int memoryProcessNum = 0;//在内存中的进程个数
    for (int i = 0; i < pcb.length; i++) {
        if (pcb[i].status.equals("Waiting")
            || pcb[i].status.equals("Running")
            || pcb[i].status.equals("Blocking")) {
            memoryProcessNum++;
        }
    }
    if (memoryProcessNum < MAX_OCCURS) {
       // 如果内存中程序数少于最大并发数，从后备队列找到最先到达的进
        程调入内存
        for (int p = 0; p < pcb.length
            && memoryProcessNum < MAX_OCCURS; p++) {
            if (pcb[p].status.equals("U") && allTime >= pcb[p].arriveTime) {
                int address = applyMemory(p);
                if (address != -2) {//分配内存成功才修改状态
                    pcb[p].address = address;
                    pcb[p].status = "Waiting";
                    memoryProcessNum++;
                }
            }
        }
    }
  }


/**
 * 调度方法
 * @throws InterruptedException
 */
public void dispatch() throws InterruptedException {
    if (isFinishALLProcess()) {
        return;
    }

    long now = System.currentTimeMillis();
    double passedTime = (now - lastTime) / 1000.0; //距离上次调度经过的时间
    double allTime = (now - beginTime) / 1000.0;
```

```
// 从后备队列调入进程
getNextOnDisk(allTime);

if (runningIndex != -1) {
	double oldhasUsedTime = pcb[runningIndex].hasUsedTime;

	if (passedTime >= timeSlice) {
		if (oldhasUsedTime + passedTime >= pcb[runningIndex].needTime) {
				pcb[runningIndex].hasUsedTime =
 pcb[runningIndex].needTime;
			pcb[runningIndex].status = "Finish";
			releaseMemory(pcb[runningIndex].address);
		} else {
			pcb[runningIndex].hasUsedTime = oldhasUsedTime + timeSlice;
			pcb[runningIndex].status = "Waiting";
		}
		// 从就绪队列中选择下一个进程
		int next = getNextWaiting(runningIndex);
		if (next != -1) {
			pcb[next].status = "Running";
			runningIndex = next;
			print();
		}
	}

} else {
	pcb[0].status = "Running";
	runningIndex = 0;
	print();
}

// 随机阻塞进程
Random random = new Random();
int randomNum = random.nextInt(3);

// 符合此条件的进程将被阻塞
if (randomNum == 2 && (blocked == 0 ) ) {
	blockIndex = runningIndex;
	pcb[runningIndex].status = "Blocking";
	blocked = 1;
	runningIndex = getNextWaiting(runningIndex);
}
// 唤醒进程
if (blocked == 1) {
```

```
            long time = System.currentTimeMillis();
            double intervalTime = (time - lastTime) / 1000.0;
            if (runningIndex == -1) {
                // 只剩下最后一个阻塞进程未执行完，则该进程休眠 1 秒后唤醒
                Thread.sleep(1000);
                runningIndex = blockIndex;
                pcb[runningIndex].status = "Waiting";
                blocked = 0;
            } else if (intervalTime > 2.0) {
                // 有进程属于阻塞状态时，满足阻塞时间大于 2s 则唤醒进程
                if (blockIndex != -1) {
                    pcb[blockIndex].status = "Waiting";
                    blocked = 0;
                }
            }
        }
    }
```

// 界面输出函数

```
    public void print() {
        System.out.println("\n-------------------------------------------------------------------------------");
        System.out.println("分区说明表11(Busy:占用 Free:空闲): ");
        System.out.println("起址(K)  长度 (KB)     状态");
        for (MemoryBlock memoryBlock : memoryBlockList) {
            System.out.printf("%-8d  %-10d  %-6s\n", memoryBlock.address, memoryBlock.length, memoryBlock.status);
        }

        // 打印后备队列中的作业
        System.out.println("\n后备队列中的作业:");
        for (int i = 0; i < PROCESS_NUM; i++) {
            if (pcb[i].status == "U") {
                System.out.print(pcb[i].name + "\t");
            }
        }

        System.out.println("\n当前各进程PCB信息: ");
        System.out.printf("进程名  到达时间/s  需要时间/s  已用时间/s  所需内存/KB      内存起址      进程状态\n");
        // 打印就绪队列、运行中、阻塞状态、完成调度这几个状态的进程
        for (int i = 0; i < PROCESS_NUM; i++) {
            if (pcb[i].status != "U") {
                System.out.printf("%-6s  %-10s  %-10s  %-10s  %-11d  %-8d  %-8s\n", pcb[i].name, d2s(pcb[i].arriveTime)
                        , d2s(pcb[i].needTime), d2s(pcb[i].hasUsedTime), pcb[i].needMemory, pcb[i].address, pcb[i].status);
            }
        }


        System.out.println("-------------------------------------------------------------------------------");
        // 更新上次调度时间
        lastTime = System.currentTimeMillis();
    }
```

6. 运行结果截图(内容太多，仅展示关键部分)：

初始的分区说明表（1024KB 的内存）：

```
----------------------------------------
内存分区表(Busy:占用 Free:空闲)：
起址(K)   长度（KB）      状态
0          1024          Free
```

开始时，后备队列中的十个作业：

```
后备队列中的作业:
P1   P2   P3   P4   P5   P6   P7   P8   P9   P10
```

按照要求开始进程调度，按照时间片轮转算法：

```
--------------------------------------------------------------------------------
内存分区表(Busy:占用 Free:空闲)：
起址(K)   长度（KB）      状态
0          50            Busy
50         974           Free

后备队列中的作业:
P2   P10 P9   P7   P6   P8   P3   P4   P5
当前各进程PCB信息：
进程名   到达时间/s   需要时间/s   已用时间/s   所需内存/KB      内存起址      进程状态
P1       0.0          14.5         0.0          50             0          Running
--------------------------------------------------------------------------------


后备队列中的作业:
P6   P8   P3   P4   P5
当前各进程PCB信息：
进程名   到达时间/s   需要时间/s   已用时间/s   所需内存/KB      内存起址      进程状态
P1       0.0          14.5         2.0          50             0          Waiting
P2       1.1          12.1         0.0          142            50         Blocking
P10      1.5          15.6         2.0          112            192        Waiting
P9       2.7          13.6         0.0          91             304        Running
P7       2.7          12.9         0.0          50             395        Waiting
--------------------------------------------------------------------------------
```

内存分区表(Busy:占用 Free:空闲)：

起址(K)	长度（KB）	状态
0	50	Busy
50	142	Busy
192	112	Busy
304	91	Busy
395	50	Busy
445	579	Free

后备队列中的作业:

P6 P8 P3 P4 P5

当前各进程PCB信息：

进程名	到达时间/s	需要时间/s	已用时间/s	所需内存/KB	内存起址	进程状态
P1	0.0	14.5	2.0	50	0	Waiting
P2	1.1	12.1	0.0	142	50	Blocking
P10	1.5	15.6	2.0	112	192	Waiting
P9	2.7	13.6	2.0	91	304	Waiting
P7	2.7	12.9	0.0	50	395	Running

内存分区表(Busy:占用 Free:空闲)：

起址(K)	长度（KB）	状态
0	50	Busy
50	142	Busy
192	112	Busy
304	91	Busy
395	50	Busy
445	579	Free

后备队列中的作业:

P6 P8 P3 P4 P5

当前各进程PCB信息：

进程名	到达时间/s	需要时间/s	已用时间/s	所需内存/KB	内存起址	进程状态
P1	0.0	14.5	2.0	50	0	Running
P2	1.1	12.1	0.0	142	50	Blocking
P10	1.5	15.6	2.0	112	192	Waiting
P9	2.7	13.6	2.0	91	304	Waiting
P7	2.7	12.9	2.0	50	395	Waiting

此时，已经运行完一些进程了：

内存分区表(Busy:占用 Free:空闲)：

起址(K)	长度（KB）	状态
0	50	Busy
50	142	Busy
192	112	Busy
304	141	Free
445	92	Busy
537	487	Free

后备队列中的作业：

P8 P3 P4 P5

当前各进程PCB信息：

进程名	到达时间/s	需要时间/s	已用时间/s	所需内存/KB	内存起址	进程状态
P1	0.0	14.5	14.0	50	0	Waiting
P2	1.1	12.1	0.0	142	50	Blocking
P10	1.5	15.6	14.0	112	192	Waiting
P9	2.7	13.6	13.6	91	304	Finish
P7	2.7	12.9	12.9	50	395	Finish
P6	3.1	15.0	0.0	92	445	Running

内存分区表(Busy:占用 Free:空闲)：

起址(K)	长度（KB）	状态
0	46	Busy
46	4	Free
50	142	Busy
192	89	Busy
281	23	Free
304	85	Busy
389	56	Free
445	92	Busy
537	487	Free

后备队列中的作业：

P5

当前各进程PCB信息：

进程名	到达时间/s	需要时间/s	已用时间/s	所需内存/KB	内存起址	进程状态
P1	0.0	14.5	14.5	50	0	Finish
P2	1.1	12.1	0.0	142	50	Blocking
P10	1.5	15.6	15.6	112	192	Finish
P9	2.7	13.6	13.6	91	304	Finish
P7	2.7	12.9	12.9	50	395	Finish
P6	3.1	15.0	6.0	92	445	Waiting
P8	3.1	13.4	6.0	85	304	Waiting
P3	4.0	15.6	2.0	46	0	Running
P4	4.9	15.0	2.0	89	192	Waiting

直至所有进程运行完成：

内存分区表(Busy:占用 Free:空闲)：

起址(K)	长度（KB）	状态
0	1024	Free

后备队列中的作业：

当前各进程PCB信息：

进程名	到达时间/s	需要时间/s	已用时间/s	所需内存/KB	内存起址	进程状态
P1	0.0	14.5	14.5	50	0	Finish
P2	1.1	12.1	12.1	142	50	Finish
P10	1.5	15.6	15.6	112	192	Finish
P9	2.7	13.6	13.6	91	304	Finish
P7	2.7	12.9	12.9	50	395	Finish
P6	3.1	15.0	15.0	92	445	Finish
P8	3.1	13.4	13.4	85	304	Finish
P3	4.0	15.6	15.6	46	0	Finish
P4	4.9	15.0	15.0	89	192	Finish
P5	4.9	15.3	15.3	46	281	Finish

---

* * * * * * * * 所有进程运行结束 * * * * * * * *

7.运行结果的分析：

第一张图是初始时候的内存分区说明表，共 1024KB，都是空闲状态。

第二张图是随机生成的十个作业。第三张图开始是进程调度，此时只有 P1 到达，因此先调入内存。运行到第四张图时，P1、P2、P10、P9、P7 已经被进入内存，剩下 P3、P4、P5、P6、P8 在后备队列中。结合第四第五张图看，可以看出，进程调度算法是时间片轮转算法，每个时间片为 2。

在同一张图片中，结合内存分区表来看，可以看到，内存分区表中状态为 Busy 的块和 PCB 信息相符合。PCB 表中的 运行中（Running）、就绪（Waiting）、阻塞（Blocking）状态的进程刚好和内存分区表中的 Busy 状态内存块一一对应。因为最多同时调入五个作业进内存，因此，内存分区表中也最多只有五个处于 Busy 状态的内存块。

当某一个进程执行完成，就会释放内存，如果释放的内存块的前后内存块也是空闲的，则进行合并，从最后一张图可以看到，最终虽有进程都执行完毕后，内存分区表最终也回到了最初的时候，只有一个 1024KB 的空闲块。

随机阻塞进程，从第四张图片开始看，会看到有进程属于阻塞状态，当阻塞到达一定的时间后会被唤醒，最终全部执行完成。

8.心得体会：

之前的实验已经写过了进程调度的四种算法，而进程调度的算法也适合于作业调度。因此这里把作业和进程均用 PCB 类来表，PCB 类被调入内存之前则是作业，当作业被调入内存之后就变为进程控制块 PCB。根据之前的实验，将这两个算法有机结合起来便可。其中，因为内存中最多同时放入五个作业，因此内存中的就绪状态、运行状态、阻塞状态这三个状态的进程数目加起来最多只能等于 5 个。与之前的实验不

同的是，本次课设里面，在进程的状态中添加了一个阻塞（Blocking）状态，在运行过程中，随机阻塞进程，待满足一定条件后再唤醒进程。同时，阻塞进程这里会产生一个边界问题，就是当内存中仅剩这一个进程了，然后这个进程又被阻塞了，这个时候就要做特殊处理了，否则整个应用程序就结束了。此时，我是将这个进程休眠了 1s，然后再唤醒该进程继续执行。

本次课设难点在在于如何将作业调度、进程调度、内存分配如何有机地结合来，虽然这个过程比较难，但是通过一步一步地分析和调试，最后也如期完成了。通过亲自动手实现，加深了对操作系统作业调度和进程调度的理解，进一步掌握了操作系统的设计、分析、实现的流程，总的来说，收获非常大。